Information for Developers
Using the SaveAsXML Plug-in

July 27, 2005
# Contents

**Preface**  ................................................................. 5  
  Introduction  .......................................................... 5  
  Other Useful Documentation  ......................................... 5  
  Conventions Used in This Book  ..................................... 5  

**Creating or Modifying Mapping Tables**  .......................... 7  
  Overview of the SaveAsXML Process  ............................... 7  
  Sample Mapping Table  ................................................ 7  
  About the Sample  ..................................................... 9  
    The Root node  ...................................................... 9  
    The Emit-string directive  ......................................... 9  
    The Walk-structure directive  .................................... 10  
    The Define-event-list directive  ................................. 10  
    The Define-proc-list directive  .................................. 13  
  Editing Mapping Tables  ............................................. 14  
    Editing Mapping Tables in FrameMaker+SGML 6.0  ............. 14  
    Guidelines for Editors other than FrameMaker+SGML 6.0 ....... 15  

**Mapping Table Elements Reference**  ............................. 17  
  Root  .............................................................................. 17  
  Walk-layout  .............................................................. 18  
  Walk-metadata  ............................................................ 18  
  Emit-all-metadata  ...................................................... 19  
  Walk-structure  ........................................................... 20  
  Walk-cached-property-sets  .......................................... 20  
  Walk-children  ........................................................... 21  
  Define-event-list  ....................................................... 21  
  Call-event-list  .......................................................... 22  
  Event  ............................................................................. 22  
  Define-proc-list  ......................................................... 24  
  Call-proc-list  ............................................................ 24  
  Proc-var  ....................................................................... 25  
  Evaluate-var  ............................................................... 28  
  Walk-proplist  ............................................................. 31  
  Proc-property  ............................................................. 31
**Introduction**

SaveAsXML is a plug-in for Adobe® Acrobat® 7.0 which extends the “Save as type” choices in the SaveAs dialog to allow a Tagged PDF document to be saved as a number of XML, HTML, or similar text-based formats.

Mapping Tables are used to control the conversion process for the SaveAsXML feature. The Mapping Tables are a script of hierarchically-organized directives written in a custom language defined in XML syntax. This allows developers to create custom Mapping Tables for formats other than those provided in this package. This document provides an overview of that language.

**Other Useful Documentation**

You will find it helpful to be familiar with the Acrobat API and Portable Document Format (PDF). The following technical notes, available with the Acrobat SDK, provide this information. Visit [http://partners.adobe.com/asn](http://partners.adobe.com/asn) to find the books you need.

*Acrobat SDK User’s Guide* provides an overview of the Acrobat SDK and the supporting documentation.

*Acrobat and PDF Library API Reference* contains the method prototypes and details on arguments.

*PDF Reference*, Version 1.6. Provides a description of the PDF file format, as well as suggestions for producing efficient PDF files. It is intended for application developers who wish to produce PDF files directly.

---

**Conventions Used in This Book**

The Acrobat documentation uses text styles according to the following conventions.

<table>
<thead>
<tr>
<th>Font</th>
<th>Used for</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>monospaced</td>
<td>Paths and filenames</td>
<td>C:\templates\mytmpl.fm</td>
</tr>
<tr>
<td></td>
<td>Code examples set off from plain text</td>
<td>These are variable declarations: AVMenu commandMenu,helpMenu;</td>
</tr>
<tr>
<td>Font</td>
<td>Used for</td>
<td>Examples</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>monospaced bold</td>
<td>Code items within plain text</td>
<td>The <code>GetExtensionID</code> method ...</td>
</tr>
<tr>
<td></td>
<td>Parameter names and literal values in reference documents</td>
<td>The enumeration terminates if <code>proc</code> returns <code>false</code>.</td>
</tr>
<tr>
<td>monospaced italic</td>
<td>Pseudocode</td>
<td><code>ACCB1</code> <code>void</code> <code>ACCB2</code> <code>ExeProc</code>(void) { <code>do</code> <code>something</code> }</td>
</tr>
<tr>
<td></td>
<td>Placeholders in code examples</td>
<td><code>AFSimple_Calculate(cFunction, cFields)</code></td>
</tr>
<tr>
<td>blue</td>
<td>Live links to Web pages</td>
<td>The Acrobat Solutions Network URL is: <code>http://partners.adobe.com/asn/</code></td>
</tr>
<tr>
<td></td>
<td>Live links to sections within this document</td>
<td>See Using the SDK.</td>
</tr>
<tr>
<td></td>
<td>Live links to code items within this document</td>
<td>Test whether an <code>ASAtom</code> exists.</td>
</tr>
<tr>
<td>bold</td>
<td>PostScript language and PDF operators, keywords, dictionary key names</td>
<td>The <code>setpagedevice</code> operator</td>
</tr>
<tr>
<td></td>
<td>User interface names</td>
<td>The <code>File</code> menu</td>
</tr>
<tr>
<td>italic</td>
<td>Document titles that are not live links</td>
<td><code>Acrobat and PDF Library API Overview</code></td>
</tr>
<tr>
<td></td>
<td>New terms</td>
<td><code>User space</code> specifies coordinates for...</td>
</tr>
<tr>
<td></td>
<td>PostScript variables</td>
<td><code>filename deletefile</code></td>
</tr>
</tbody>
</table>
Creating or Modifying Mapping Tables

Overview of the SaveAsXML Process

When the SaveAsXML plug-in registers itself with Acrobat 7.0, it inspects the set of XML files in the MappingTables folder to determine the number of conversion services that are available.

- The MappingTables folder must be found inside SaveAsXML folder which is at the same level as SaveAsXML.ap.
- Files in this folder are the only ones that are inspected as potential conversion services supported by the plug-in.
- This folder may not contain any files with the .xml extension that are not Mapping Tables.

If the registration process finds the Root element and its menu-name attribute, which may be a string or a predefined identifier, it adds the menu-name to the list of file format choices available in the SaveAs dialog. (The menu-name must be unique, or the user may be confused by similarly identified entries among the SaveAs dialog's file formats.)

The following Sample Mapping Table (simplified and incomplete) demonstrates the basic operation of the SaveAsXML processing. The complete sample is shown first, followed by an annotated version with explanations.

For more complete examples of the usage of these directives, see the Mapping Tables distributed with SaveAsXML. Every directive that is currently supported has been used in one or more of these tables.

The following section, Mapping Table Elements Reference, provides details of the full list of directives and their attributes.

Sample Mapping Table

```xml
<Root File-format = "Xml-1-00" Menu-name = "Sample Mapping Table"
Mac-creator = "MSIE" Mac-type = "TEXT" Win-suffix = "xml"
Encode-out = "Utf-8-out">
  <Emit-string ... >&lt;XML-Doc&gt;</Emit-string>
  <Walk-structure Use-event-list = "Block-events"></Walk-structure>
  <Emit-string ...>&lt;/XML-Doc&gt;</Emit-string>
  <Define-event-list Name = "Block-events">
    <Event Inf-type = "Struct-elem" Name-type = "Structure-role"
      Node-name = "Div" Alternate-name = "-none-"
      Node-content = "Has-kids" Event-class = "Enter">
      <Emit-string ...>&lt;Div</Emit-string>
    </Event>
  </Define-event-list>
</Root>
```
Creating or Modifying Mapping Tables

Sample Mapping Table

<Call-proc-list Name = "Block-attributes"></Call-proc-list>
<Emit-string ...&gt;</Emit-string>
<Walk-children Use-event-list = "Inline-events"></Walk-children>
</Event>
<Event Inf-type = "Struct-elem" Name-type = "Structure-role"
Node-name = "Div" Alternate-name = ":-none-
Node-content = "Has-kids" Event-class = "Exit">
<Emit-string ...&gt;&lt;/Div&gt;</Emit-string>
</Event>
<Event Inf-type = "Struct-elem" Name-type = "Structure-role"
Node-name = "Div" Alternate-name = ":-none-
Node-content = "Empty" Event-class = "Enter">
<Emit-string ...&lt;/Div</Emit-string>
<Call-proc-list Name = "Block-attributes"></Call-proc-list>
<Emit-string ...&gt;</Emit-string>
</Event>

<Define-event-list Name = "Inline-events">
<Define-event-list Name = "Inline-events">
<Event Inf-type = "Struct-elem" Name-type = "Structure-role"
Node-name = "Span" Alternate-name = ":-none-
Node-content = "Has-kids" Event-class = "Enter">
<Emit-string ...&lt;Span</Emit-string>
<Call-proc-list Name = "Span-attributes"></Call-proc-list>
<Emit-string ...&gt;</Emit-string>
<Walk-children Use-event-list = "Inline-events"></Walk-children>
</Event>
<Event Inf-type = "Struct-elem" Name-type = "Structure-role"
Node-name = "Span" Alternate-name = ":-none-
Node-content = "Has-kids" Event-class = "Exit">
<Emit-string ...&lt;/Span&gt;</Emit-string>
</Event>
<Event Inf-type = "Struct-elem" Name-type = "Structure-role"
Node-name = "Span" Alternate-name = ":-none-
Node-content = "Empty" Event-class = "Enter">
<Emit-string ...&lt;Span</Emit-string>
<Call-proc-list Name = "Span-attributes"></Call-proc-list>
<Emit-string ...&gt;</Emit-string>
<Walk-children Use-event-list = "Inline-events"></Walk-children>
</Event>
<br />
</Define-event-list>
<br />
<Define-proc-list Name = "Block-attributes">
<br />
<Proc-var Pdf-var = "Alt" Owner = "Structelem" Type = "String"
Has-enum = "No-enum" Inherit = "Not-inherited" Default = ":-none-
Condition = "Has-value">
<br />
<br />
<br />
</Proc-var>
</Define-proc-list>

Information for Developers Using the SaveAsXML Plug-in
Creating or Modifying Mapping Tables

About the Sample

Once the user selects an applicable file format in the SaveAs dialog, the dialog handler activates the SaveAsXML plug-in. The plug-in reads the associated Mapping Table and converts it to a binary in-memory format, which it uses to control the processing of the current TaggedPDF document.

The Root node

Processing begins with the root node of the Mapping Table and generally proceeds as a pre-order hierarchical traversal of the control nodes.

```
<Root File-format = "Xml-1-00" Menu-name = "Sample Mapping Table"
      Mac-creator = "MSIE" Mac-type = "TEXT" Win-suffix = "xml"
      Encode-out = "Utf-8-out">
```

In processing the **Root** node of the Mapping Table, the SaveAsXML processor opens the output file using the filepath and name of the PDF document to be saved, replacing the file suffix with that specified by the **Win-suffix** attribute in this node. On the Macintosh, the **Mac-creator** and **Mac-type** are also used to open the output file. The remaining attributes in the **Root** node are available to the SaveAsXML processor and are internally used to control or optimize the conversion.

The Emit-string directive

```
<Emit-string ... >&lt;XML-Doc&gt;</Emit-string>
```

The **Emit-string** directive causes its content to be translated to the output encoding specified in the **Encode-out** attribute of the **Root** node, then emits the converted data to the output file. In this case, it issues the start tag for the document: `<XML-Doc>

**NOTE:** For clarity, the additional attributes of the **Emit-string** directive have been omitted in this sample.
Here, as in any Mapping Table directive:

- `&lt;` represents the less-than (<) character.
- `&gt;` represents the greater-than (>) character.
- `&amp;` represents the ampersand (&) character.

### The Walk-structure directive

```xml
<Walk-structure Use-event-list = "Block-events"></Walk-structure>
```

The `Walk-structure` directive causes the SaveAsXML processor to walk the first level `Structural Elements` (Kids array of the `StructRoot`) of the Tagged PDF document to be saved. (See **The Walk-children directive**.)

Structural Elements are traversed in the order found in the *Logical Structure Tree*. An event is generated on entering and on exiting each Structural Element. The event-list specified by the `Use-event-list` attribute of the `Walk-structure` directive is searched for a matching `Event` directive (See **The Define-event-list directive**).

- If a match is found, the directives within that `Event` directive are processed (which may include the recursive processing of children of the current Structural Element via a `Walk-children` directive). Searching of the event-list is terminated and the next event is generated.

- If no match is found (or when processing is completed on the matching `Event` directive) then the next event is generated.

Processing continues until all first-level Structural Elements (Kids of the `StructRoot`) have been traversed, then the directive following the `Walk-structure` directive is processed. In this case, it is:

```xml
<Emit-string Emit-space-after = "Emit-space-after" ...
  &lt;/XML-Doc&gt;
</Emit-string>
```

This `Emit-string` directive issues the end tag: `</XML-Doc>`. Since newlines and spaces are often modified or stripped by various XML tools, the `Emit-space-after` attribute (and the other related attributes of the `Emit-string` directive) guarantees the retention of these characters.

### The Define-event-list directive

```xml
<Define-event-list Name = "Block-events">
```

The `Define-event-list` directive is similar to a macro or subroutine definition in most programming languages. It encapsulates and names a set of event directives that are activated by a `Walk-structure`, a `Walk-children`, or a `Call-event-list` directive having a corresponding name in its `Use-event-list` attribute.
The Event directive

```xml
<Event Inf-type = "Struct-elem" Name-type = "Structure-role"
Node-name = "Div" Alternate-name = "-none-"
Node-content = "Has-kids" Event-class = "Enter"/>
```

The Event directive includes a set of attributes that are used to determine if the directives within it are to be processed. For a complete description of these attributes, see the full specification of this directive in the next section of this document. The directive above is activated by a entering (either from a parent element or from the prior peer element) a Structural Element (Inf-type = "Struct-elem"), when the element is role-mapped (Name-type = "Structure-role") to "Div" and the element has children (see the 2nd event directive below for an element that has no children).

When an Event directive is activated, the directives within it (before its </Event>tag) are processed. In this case:

```xml
<Emit-string ...>&lt;Div</Emit-string>
```

This issues the "<Div " portion of the output element's start-tag.

The Call-proc-list directive

```xml
<Call-proc-list Name = "Block-attributes"></Call-proc-list>
```

The Call-proc-list directive processes the properties associated with this Structural Element, using the processing list specified by the Name property on the Call-proc-list directive.

Although the event-list processing stops on the first match, the proc-list processing continues for every directive in the selected processing list. (The Block-attributes proc-list is described later in this example.)

```xml
<Emit-string ...>&gt;</Emit-string>
```

Issues the closing ">" on the output element's start-tag.

The Walk-children directive

```xml
<Walk-children Use-event-list = "Inline-events"></Walk-children>
```

The Walk-children directive is functionally identical to the Walk-structure directive (described earlier in this example), except that it walks the first level children of the current Structural Element.

```xml
</Event>
```

The </Event>tag indicates the end of the processing for this event. Remaining entries in this event-list follow a similar model.

The next Event included in this event-list handles events that are generated when exiting Div elements that have children. This generates the close tag on the output element.

```xml
<Event Inf-type = "Struct-elem" Name-type = "Structure-role"
Node-name = "Div" Alternate-name = "-none-"
Node-content = "Has-kids" Event-class = "Exit">
<Emit-string ...>&lt;/Div&gt;</Emit-string>
</Event>
Creating or Modifying Mapping Tables

About the Sample

The final `Event` directive included in this event-list handles events that are generated on entering an element which has no children. (Note that it does not and should not contain a `Walk-children` directive.)

```xml
<Event
  Inf-type = "Struct-elem" Name-type = "Structure-role"
  Node-name = "Div" Alternate-name = "-none-
  Node-content = "Empty" Event-class = "Enter">
  <Emit-string ...
    &lt;Div\</Emit-string>
  <Call-proc-list Name = "Block-attributes"></Call-proc-list>
  <Emit-string ...
    &gt;</Emit-string>
</Event>
</Define-event-list>
```

The `</Define-event-list>` tag ends the list of entries in the `Block-events` event-list.

The following event-list is for handling inline elements and is similar to the one above.

```xml
<Define-event-list Name = "Inline-events">
  <Event
    Inf-type = "Struct-elem" Name-type = "Structure-role"
    Node-name = "Span" Alternate-name = "-none-
    Node-content = "Has-kids" Event-class = "Enter">
    <Emit-string ...
      &lt;Span\</Emit-string>
    <Call-proc-list Name = "Span-attributes"></Call-proc-list>
    <Emit-string ...
      &gt;</Emit-string>
    <Walk-children Use-event-list = "Inline-events">
    </Walk-children>
  </Event>
  <Event
    Inf-type = "Struct-elem" Name-type = "Structure-role"
    Node-name = "Span" Alternate-name = "-none-
    Node-content = "Has-kids" Event-class = "Exit">
    <Emit-string ...
      &lt;/Span\</Emit-string>
  </Event>
  <Event
    Inf-type = "Struct-elem" Name-type = "Structure-role"
    Node-name = "Span" Alternate-name = "-none-
    Node-content = "Empty" Event-class = "Enter">
    <Emit-string ...
      &lt;Span\</Emit-string>
    <Call-proc-list Name = "Span-attributes"></Call-proc-list>
    <Emit-string ...
      &gt;</Emit-string>
  </Event>
</Define-event-list>
```

For event-lists that process Structural Elements that contains text or graphics, an `Event` entry like the following is needed. The code in the SaveAsXML plug-in that traverses the Logical Structure Tree also reports entering and exiting of the marked content containers (the wrappers around the low-level text and graphic content in the PDF page's marking stream). The labels on these nodes are hidden in the Tags view in Acrobat. (The corresponding `Event` for a `Pds-mc` element where the content is `Image` is slightly more complex. See the Mapping Tables distributed with SaveAsXML for complete examples.)

```xml
<Event
  Inf-type = "Pds-mc" Name-type = "Any" Node-name = "-none-
  Alternate-name = "-none-" Node-content = "Has-text-only"
  Event-class = "Enter">
```

Information for Developers Using the SaveAsXML Plug-in
This Event directive processes the low-level marked content containers (Inf-type = "Pds-mc") that actually contain the text (Node-content = "Has-text-only"). A corresponding exit directive is not required.

The Proc-doc-text directive

The Proc-doc-text directive converts the text from the active marked content container in the PDF page's marking stream to the output encoding specified in the Encode-out attribute of the Root node and then emits the converted data to the output file. The do-br-substitution attribute controls whether the LF character is to be converted to a <BR/> tag in the output stream, converted to a space, or discarded.

The Define-proc-list directive

The Define-proc-list directive is also a macro/subroutine similar to the Define-event-list directive. Whereas the event-list describes how to process transition events in traversing the Logical Structure Tree, the proc-list describes how to process the properties (attributes) of a Structural Element.

The Proc-var directive

The Proc-var directive searches an internal cache of the properties on the current Structural Element for the value of the property specified by its Pdf-var and Owner attributes. If inheritance is enabled, it also searches the cached properties of all ancestors of the current Structural Element for an applicable value. Once it determines if there is (or is not) a value, it then uses the remaining attributes to determine if the value should be processed. If it determines it should be processed, then the directives contained in this Proc-var directive are processed.

The Proc-string directive

The Proc-string directive causes the string selected by the containing Proc-var directive to be translated to the output encoding specified in the Encode-out attribute of the Root node and then emits the converted data to the output file.

The </Define-proc-list> tag indicates the end of this proc-list.
The following proc-list has a similar organization to the one, above, for Block-attributes.

```xml
<Define-proc-list Name = "Span-attributes">
    <Proc-var Pdf-var = "ActualText" Owner = "Structelem"
                Type = "String" Has-enum = "No-enum"
                Inherit = "Not-inherited" Default = "-none-"
                Condition = "Always">
        <Emit-string ...>actual-text="</Emit-string>
        <Proc-string></Proc-string>
        <Emit-string ..."></Emit-string>
    </Proc-var>
</Define-proc-list>
</Root>
```

The `</Root>` tag is the last line of a Mapping Table file. It indicated the end of the `Root` directive.

---

**Editing Mapping Tables**

You can edit the .xml versions of the Mapping Tables in any XML (or SGML) editor. The files were created using FrameMaker+SGML 6.0, and detailed instructions for that editor are included below, followed by general instructions for using another editor.

**Editing Mapping Tables in FrameMaker+SGML 6.0.**

To edit/modify the Mapping Tables in FrameMaker+SGML 6.0:

- Copy all the files in the SaveAsXML/DeveloperInfo folder and all the mapping tables from the SaveAsXML/MappingTables folder to a single folder on your machine. Be sure to include:
  - `sgmlapps.fm`
  - `sgml.dec`
  - `MappingTable.edd.fm`
  - `MappingTable.dtd`
  - `MappingTable.fm` and
  - the `___.xml` (or `___.fm`) file for the conversion you wish to edit.

- Open the `sgmlapps.fm` file:
  - Select the menu command:
    - `File => Developer Tools => Reread SGML Application File`
  - Close this file but do not exit FrameMaker; the reread `sgmlapps.fm` file remains valid only for the current session.

- Choose the SGML Application
Creating or Modifying Mapping Tables

Editing Mapping Tables

– Select the menu command:

File => Set SGML Application

– Choose Mapping Table from the pulldown in the dialog.

– Select Set to close the dialog.

● Open the .xml version of the Mapping Table file you wish to edit (note that
FrameMaker changes the file extension to .fm, do not change it back to .xml, see
instructions later in this section):

– Make the necessary changes.

– Select the menu command:

Element => Validate then click "Start Validating"

Correct what is necessary until you get a "Document is valid" response.

● Select the menu command:

File => Save to save the .fm version of the file.

● Select the menu command:

File => SaveAs

– In the save as type field, choose SGML (Note: Do NOT choose XML)

– BE SURE TO CHANGE THE FILE SUFFIX TO .xml

– Click "Save"

– Copy the .xml file to the Plug-Ins/SaveAsXML/MappingTables folder.

Guidelines for Editors other than FrameMaker+SGML 6.0

The DTD is included in the DeveloperInfo folder is an SGML syntax DTD. To convert it to XML
syntax, remove the " - -" (space-hyphen-space-hyphen, which indicates start tag and end
tag are required) from each ELEMENT directive.

It may be necessary to modify the file path:

"D:\Adobe Docs\AcroStructure\MappingTables\MappingTable.dtd"
in the DOCTYPE directive to point to your local copy of the DTD.

NOTE: The SaveAsXML processor requires that the Mapping Tables must be valid in
accordance with this DTD or Acrobat may crash during the SaveAs operation.
Mapping Table Elements Reference

This section provides complete details of all Mapping Table directives and their attributes.

**Root**

This is the root node of a Mapping Table. Its attributes specify the name of the filter to appear in the menu and information necessary to properly generate the output file name and type information.

**DTD Content Rule**

(Comment | Emit-string | Define-event-list | Define-proc-list | Walk-metadata | Emit-all-metadata | Walk-cached-property-sets | Walk-structure | Walk-layout)+

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File-format</strong></td>
<td>Choice</td>
<td>Required—Internal name that describes the format of the output file (must be unique). The following formats are provided at release:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Html-3-02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Html-4-01-with-css-1-00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Xml-1-00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Plain Text</td>
</tr>
<tr>
<td><strong>Menu-name</strong></td>
<td>String or Identifier</td>
<td>Required—the text string describing the file format that appears in the SaveAs dialog’s pulldown menu. The following predefined identifiers, which provide localized menu name strings, may be used in place of a string:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$IDS_HTML_3_2_MENU_NAME - localized string &quot;HTML 3.2&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$IDS_HTML_4_01_CSS_1_0_MENU_NAME - localized string &quot;HTML 4.01 with CSS 1.0&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$IDS_XML_1_0_MENU_NAME - localized string &quot;XML 1.0&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$IDS_PLAIN_TEXT_MENU_NAME - localized string &quot;Text (Plain)&quot;</td>
</tr>
<tr>
<td><strong>Mac-creator</strong></td>
<td>String</td>
<td>Required—the file creator field for a Macintosh file.</td>
</tr>
<tr>
<td><strong>Mac-type</strong></td>
<td>String</td>
<td>Required—the file type field for a Macintosh file.</td>
</tr>
<tr>
<td><strong>Win-suffix</strong></td>
<td>String</td>
<td>Required—the 3 letter filetype suffix for the Windows environment. Also used on Macintosh files.</td>
</tr>
</tbody>
</table>
## Walk-layout

**THIS DIRECTIVE IS NOT SUPPORTED IN THIS VERSION OF SaveAsXML.**

## Walk-metadata

Directs the SaveAs processor to walk the DocInfo metadata portion of the PDF document.

**DTD Content Rule**

*Void?*

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Use-proc-list</code></td>
<td>String</td>
<td>Required—The name of an event processing list (see <code>&lt;define-proc-list&gt;</code>) to be used to process the attributes found by walking the metadata portion of the document.</td>
</tr>
</tbody>
</table>
**Emit-all-metadata**

Copies the full set of XAP metadata to the output file.

**DTD Content Rule**

Void?

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Emit-space-before  | Choice | Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  ● **Emit-space-before**: Emit a space before emitting any content text.  
  ● **No-space-before**: Do not emit a space before emitting any content text. |
| Emit-space-after    | Choice | Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  ● **Emit-space-after**: Emit a space after emitting any content text.  
  ● **No-space-after**: Do not emit a space after emitting any content text. |
| Emit-newline-before | Choice | Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  ● **Emit-newline-before**: Emit a newline before emitting any content text.  
  ● **No-newline-before**: Do not emit a newline before emitting any content text. |
| Emit-newline-after  | Choice | Required—Since XML strips the first/last space and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  ● **Emit-newline-after**: Emit a newline after emitting any content text.  
  ● **No-newline-after**: Do not emit a newline after emitting any content text. |
**Walk-structure**

Directs the SaveAs processor to walk the Logical Structure Tree and associated content of the PDF document.

**DTD Content Rule**

Void?

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use-event-list</td>
<td>String</td>
<td>Required—The name of an event processing list (see <code>&lt;define-event-list&gt;</code>) to be used to process the events generated by walking the structure tree (PDF Logical Structure) of the document.</td>
</tr>
</tbody>
</table>

**Walk-cached-property-sets**

Directs the SaveAs processor to construct a stylesheet cache and walk the stylesheet data.

**DTD Content Rule**

Void?

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use-event-list</td>
<td>String</td>
<td>Required—The name of an event processing list (see <code>&lt;define-event-list&gt;</code>) to be used to process the events generated by walking the stylesheet data (ClassMap and class information) of the document.</td>
</tr>
</tbody>
</table>
**Walk-children**

Directs the SaveAs processor to walk the kids list of the current Structural Element.

**DTD Content Rule**

Void?

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use-event-list</td>
<td>String</td>
<td>Required—The name of an event processing list (see <code>&lt;define-event-list&gt;</code> to be used to process the events generated by walking the first-level children of the current Structural Element.</td>
</tr>
</tbody>
</table>

**Define-event-list**

Event-lists and proc-lists, like macros, allow the user to define a series of processing directives which may be used in multiple locations within the SaveAs Mapping Table.

- Event-lists govern the selection and processing of elements in the layout, metadata, logical structure, or stylesheet trees.
- Proc-lists govern the processing of attributes/properties associated with a given event/Structural Element. (See **Define-proc-list**.)

**DTD Content Rule**

( Comment | Event | Call-event-list)+

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Required—The name to be applied to the event processing list being defined by this element. This is referenced in the <code>&lt;Walk-*&gt;</code> elements via the &quot;Use-event-list&quot; attribute. The name must be unique across all Define-event-list elements within a given Mapping Table file.</td>
</tr>
</tbody>
</table>
### Call-event-list

Identical to a macro call, inserts the named event-list at this point in the Mapping Table.

**DTD Content Rule**

Void?

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Required—The name of a event list (see <code>&lt;Define-event-list&gt;</code>) to be included at this point in the current event list.</td>
</tr>
</tbody>
</table>

### Event

Governs the processing of a node in the layout, logical-structure, metadata, or stylesheet trees. Specifies the processing that is to be performed on entering or exiting the named node.

**DTD Content Rule**


**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Node-type  | Choice | Required—The Node-name attribute is matched against either the $S$ key of the StructElem (Structure-user-label) or against the result of processing that key via the RoleMap (Structure-role). One of:  
  - **Any**: Attempt match on Structure-user-label then on Structure-role. Also used for matching within metadata an stylesheet construction.  
  - **Structure-role**: Compare Node-name to the result of processing the StructElem's $S$ key via the RoleMap.  
  - **Structure-user-label**: Compare Node-name to the StructElem's $S$ key. |
| Node-name  | String | Required—Name of the element/role to match, in order to select this event descriptor for processing. |
## Mapping Table Elements Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node-content</strong></td>
<td>Choice</td>
<td>Required. One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Empty</strong>: Node has no children or direct content.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Has-text-only</strong>: Node has only text content (no other elements).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Has-kids</strong>: Node has child elements (including possible text-only spans).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Graphic</strong>: Node contains (vector) graphic data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Image</strong>: Node contains bitmapped image data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Other</strong>: Node is something other than those listed above.</td>
</tr>
<tr>
<td><strong>Event-class</strong></td>
<td>Choice</td>
<td>Required—Identifies which transition into or out of the node is to be processed using this event description. One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Enter</strong>: Node is being entered from either parent or peer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Enter-from-parent</strong>: Node is being entered from parent, but not from peer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Enter-from-peer</strong>: Node is being entered from peer, but not from parent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Exit</strong>: Node is being exited to either parent or to peer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Exit-to-parent</strong>: Node is being exited to parent, but not to peer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Exit-to-peer</strong>: Node is being exited to peer, but not to parent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Begin-children</strong>: Node is being exited to begin processing its children.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>End-children</strong>: Node is being re-entered after processing its children.</td>
</tr>
</tbody>
</table>
Define-proc-list

Proc-lists and event-lists, like macros, allow the user to define a series of processing directives which may be used in multiple locations within the SaveAs Mapping Table.

- Proc-lists govern the processing of attributes/properties associated with a given event/Structural Element.
- Event-lists govern the selection and processing of elements in the layout, metadata, logical structure, or stylesheet trees. (See Define-event-list.)

**DTD Content Rule**

```
(Comment | Proc-var | Walk-proplist | Call-proc-list)+
```

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Required—the name to be applied to the variable processing list being defined by this element. This is referenced in the <code>&lt;Call-proc-list&gt;</code> element via its Name attribute. The name must be unique across all Define-proc-list elements within a given Mapping Table file.</td>
</tr>
</tbody>
</table>

Call-proc-list

Identical to a macro call, inserts the named proc-list at this point in the Mapping Table.

**DTD Content Rule**

```
Void?
```

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Required—the name of a variable processing list (see <code>&lt;define-proc-list&gt;</code> to be included at this point in the current event or proc-list.</td>
</tr>
</tbody>
</table>
**Proc-var**

Specifies the formatting/conversion of the named attribute/property (PDF-variable). This directive also caches the data value and type of the value specified for use by various processing directives within this element.

**DTD Content Rule**

```
(Comment | Conditional-delimeter | Emit-string | Conditional-prefix |
Element-name | Proc-string | Proc-integer | Proc-fixed | Proc-length |
Proc-image-content | Conditional-suffix )+
```

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pdf-var</strong></td>
<td>String</td>
<td>Required—The name of a property in a given property dictionary (see Owner) to be processed/evaluated.</td>
</tr>
<tr>
<td><strong>Owner</strong></td>
<td>Choice</td>
<td>Required—The owner of the property dictionary. One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Metadata</strong>: This is a pseudo-owner for entries in the document’s metadata.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Structelem</strong>: This is a pseudo-owner for properties specified directly in the StructElem’s obj dictionary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Layout</strong>: Properties in the StructElem’s Attribute dictionary list within the dictionary owned by Layout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Link</strong>: Properties in the StructElem’s Attribute dictionary list within the dictionary owned by Link.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>List</strong>: Properties in the StructElem’s Attribute dictionary list within the dictionary owned by List.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Table</strong>: Properties in the StructElem’s Attribute dictionary list within the dictionary owned by Table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Auto-span</strong>: This is a pseudo-owner generated by the SaveAs processor for each span it synthesizes by consolidating Tj operators having common styling properties (font, size, color, etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Inline-markup</strong>: This is a pseudo-owner generated by the SaveAs processor when the following inline marking is encountered:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/Span &lt;&lt; ... &gt;&gt; BDC (abbrev.) Tj EMC</td>
</tr>
</tbody>
</table>

Information for Developers Using the SaveAsXML Plug-in
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Type    | Choice | Required—The primary PDF datatype of the property (see Has-enum for a possible secondary datatype). One of:  
  ● **Fixed**: Fixed-point number.  
  ● **Int32**: A signed integer.  
  ● **Atom**: A PDF key (/XYZ).  
  ● **String**: A PDF string.  
  ● **Color**: An RGB color (array of 3 Fixed values)  
  ● **BBox**: A bounding box (array of 4 Fixed values) |
| Inherit | Choice | Optional — Whether the property value can be inherited from a parent. One of:  
  ● **Inheritable**: This property can be inherited.  
  ● **Not-inherited**: This property can not be inherited (Default). |
| Default | String | Optional — The value to be used if the property is not found on this element (or through inheritance). This should be the same type (Fixed, Int32, Atom, String) as the property. |
| Condition | Choice | Required — Indicates whether the directives that are children of the Proc-var directive are to be executed. One of:  
  ● **Always**: Always execute the children of this Proc-var directive.  
  ● **Has-value**: Execute the children of this Proc-var directive if a value is found on this node (either explicit or Default).  
  ● **Diff-from-default-for-event**: Execute the children of this Proc-var directive if a value is found and that value differs from that specified by Default.  
  ● **Diff-from-ancestor**: Execute the children of this Proc-var directive if a value is found and that value differs from that specified by searching the inheritance tree for any ancestor.  
  ● **Diff-from-parent**: Execute the children of this Proc-var directive if a value is found and that value differs from that specified by examining the inheritance cache of the parent.  
  ● **Diff-from-predecessor**: Execute the children of this Proc-var directive if a value is found and that value differs from that specified by examining the inheritance cache of the preceding peer. |
### Condition (cont’d)

- **Diff-from-value**: Execute the children of this `Proc-var` directive if a value is found and that value differs from that specified by `Compare`. (Can be used with any type)
- **Matches-value**: Execute the children of this `Proc-var` directive if a value is found and that value matches that specified by `Compare`. (Can be used with any type)
- **Less-than-value**: Execute the children of this `Proc-var` directive if a value is found and that value is less than that specified by `Compare`. (Can only be used with: Fixed, Int32, Atom, String)
- **Less-equal-value**: Execute the children of this `Proc-var` directive if a value is found and that value is less than or equal to that specified by `Compare`. (Can only be used with: Fixed, Int32, Atom, String)
- **More-than-value**: Execute the children of this `Proc-var` directive if a value is found and that value is greater than that specified by `Compare`. (Can only be used with: Fixed, Int32, Atom, String)
- **More-equal-value**: Execute the children of this `Proc-var` directive if a value is found and that value is greater than or equal to that specified by `Compare`. (Can only be used with: Fixed, Int32, Atom, String)

### Comparison Values

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compare</strong></td>
<td>String</td>
<td>Optional—The value used to determine <code>Diff-from-value</code>, <code>Matches-value</code>, <code>Less-than-value</code>, or <code>More-than-value</code>. This should be the same type (Fixed, Int32, Atom, String) as the property.</td>
</tr>
</tbody>
</table>
Evaluate-var

Does exactly the same processing as Proc-var, except it does not make the data value available to the other contained processing directives.

DTD Content Rule


Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pdf-var</td>
<td>String</td>
<td>Required—The name of a property in a given property dictionary (see Owner) to be processed/evaluated.</td>
</tr>
<tr>
<td>Owner</td>
<td>Choice</td>
<td>Required—The owner of the property dictionary. One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Metadata: This is a pseudo-owner for entries in the document’s metadata.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● StructElem: This is a pseudo-owner for properties specified directly in the StructElem’s obj dictionary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Layout: Properties in the StructElem’s Attribute dictionary list within the dictionary owned by Layout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Link: Properties in the StructElem’s Attribute dictionary list within the dictionary owned by Link.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● List: Properties in the StructElem’s Attribute dictionary list within the dictionary owned by List.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Table: Properties in the StructElem’s Attribute dictionary list within the dictionary owned by Table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Auto-span: This is a pseudo-owner generated by the SaveAs processor for each span it synthesizes by consolidating Tj operators having common styling properties (font, size, color, etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Inline-markup: This is a pseudo-owner generated by the SaveAs processor when the following inline marking is encountered:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/Span &lt;&lt; ... &gt;&gt; BDC (abbrev.) Tj EMC</td>
</tr>
</tbody>
</table>

Information for Developers Using the SaveAsXML Plug-in
### Mapping Table Elements Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Type** | Choice | Required—The primary PDF datatype of the property (see Has-enum for a possible secondary datatype). One of:  
  - **Fixed**: Fixed-point number.  
  - **Int32**: A signed integer.  
  - **Atom**: A PDF key (/XYZ).  
  - **String**: A PDF string.  
  - **Color**: An RGB color (array of 3 Fixed values)  
  - **BBox**: A bounding box (array of 4 Fixed values)  |
| **Inherit** | Choice | Optional—Whether the property value can be inherited from a parent. One of:  
  - **Inheritable**: This property can be inherited.  
  - **Not-inherited**: This property can not be inherited (Default).  |
| **Default** | String | Optional—The value to be used if the property is not found on this element (or through inheritance). This should be the same type (Fixed, Int32, Atom, String) as the property.  |
| **Condition** | Choice | Required—Indicates whether the directives that are children of the Proc-var directive are to be executed. One of:  
  - **Always**: Always execute the children of this Proc-var directive.  
  - **Has-value**: Execute the children of this Proc-var directive if a value is found on this node (either explicit or Default).  
  - **Diff-from-default-for-event**: Execute the children of this Proc-var directive if a value is found and that value differs from that specified by Default.  
  - **Diff-from-ancestor**: Execute the children of this Proc-var directive if a value is found and that value differs from that specified by searching the inheritance tree for any ancestor.  
  - **Diff-from-parent**: Execute the children of this Proc-var directive if a value is found and that value differs from that specified by examining the inheritance cache of the parent.  
  - **Diff-from-predecessor**: Execute the children of this Proc-var directive if a value is found and that value differs from that specified by examining the inheritance cache of the preceding peer.  |
### Condition (cont’d)

- **Diff-from-value**: Execute the children of this `Proc-var` directive if a value is found and that value differs from that specified by `Compare`. (Can be used with any type)
- **Matches-value**: Execute the children of this `Proc-var` directive if a value is found and that value matches that specified by `Compare`. (Can be used with any type)
- **Less-than-value**: Execute the children of this `Proc-var` directive if a value is found and that value is less than that specified by `Compare`. (Can only be used with: Fixed, Int32, Atom, String)
- **Less-equal-value**: Execute the children of this `Proc-var` directive if a value is found and that value is less than or equal to that specified by `Compare`. (Can only be used with: Fixed, Int32, Atom, String)
- **More-than-value**: Execute the children of this `Proc-var` directive if a value is found and that value is greater than that specified by `Compare`. (Can only be used with: Fixed, Int32, Atom, String)
- **More-equal-value**: Execute the children of this `Proc-var` directive if a value is found and that value is greater than or equal to that specified by `Compare`. (Can only be used with: Fixed, Int32, Atom, String)

### Compare

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare</td>
<td>String</td>
<td>Optional—The value used to determine Diff-from-value, Matches-value, Less-than-value, or More-than-value. This should be the same type (Fixed, Int32, Atom, String) as the property.</td>
</tr>
</tbody>
</table>
Walk-proplist

Directs the SaveAs processor to walk the specified generic property list (property lists owned by XML, HTML-3.20, HTML-4.01). This is used to process arbitrary, user-supplied attributes on the current Structural Element.

**DTD Content Rule**

```
(Comment | Conditional-delimeter | Emit-string | Proc-property)+
```

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner</td>
<td>Choice</td>
<td>Required—Selects the attribute list owner. One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>●Xml</td>
</tr>
<tr>
<td></td>
<td></td>
<td>●Html-3.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>●Html-4.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>●Css-1.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>●Css-2.00</td>
</tr>
</tbody>
</table>

**Proc-property**

Processes an arbitrary property. This is similar in function to `proc-var`, except that it does not select or filter which properties are processed, but simply takes each property owned by the current owner in turn.

**DTD Content Rule**

```
(Comment | Conditional-delimeter | Emit-string | Property-name | Property-type)+
```
Property-type

Processes the data portion of an arbitrary property.

**DTD Content Rule**


**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Choice</td>
<td>Required—The primary PDF datatype of the property. One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Fixed</strong>: Fixed-point number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Int32</strong>: A signed integer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Atom</strong>: A PDF key (/XYZ).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>String</strong>: A PDF string.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>Color</strong>: An RGB color (array of 3 Fixed values)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● <strong>BBox</strong>: A bounding box (array of 4 Fixed values)</td>
</tr>
</tbody>
</table>

Property-name

Processes the name/key portion of an arbitrary property.

**DTD Content Rule**

Void?
**Element-name**

Outputs the Element-name (used in the XML output filter to generate the user-supplied element tag.

**DTD Content Rule**

Void?

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node-type</td>
<td>Choice</td>
<td>Required—Whether to get the Structural Element name to emit directly from the <code>/S</code> key of the StructElem (Structure-user-label) or from the result of processing that key via the RoleMap (Structure-role). One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Structure-role*: Use the result of processing the StructElem's <code>/S</code> key via the RoleMap.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Structure-user-label*: Use the StructElem's <code>/S</code> key.</td>
</tr>
</tbody>
</table>

**Conditional-delimeter**

Emits the contained text if this `proc-var` is not the first one to be accepted and processed after the start of an event or the first one to be processed after a `conditional-prefix` control element.

**DTD Content Rule**

`<TEXT>`

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emit-space-before</td>
<td>Choice</td>
<td>Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* Emit-space-before*: Emit a space before emitting any content text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* No-space-before*: Do not emit a space before emitting any content text.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Emit-space-after</td>
<td>Choice</td>
<td>Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Emit-space-after</strong>: Emit a space after emitting any content text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>No-space-after</strong>: Do not emit a space after emitting any content text.</td>
</tr>
<tr>
<td>Emit-newline-before</td>
<td>Choice</td>
<td>Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Emit-newline-before</strong>: Emit a newline before emitting any content text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>No-newline-before</strong>: Do not emit a newline before emitting any content text.</td>
</tr>
<tr>
<td>Emit-newline-after</td>
<td>Choice</td>
<td>Required—Since XML strips the first/last space and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Emit-newline-after</strong>: Emit a newline after emitting any content text.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>No-newline-after</strong>: Do not emit a newline after emitting any content text.</td>
</tr>
</tbody>
</table>
**Conditional-prefix**

Caches and emits the contained text if any **proc-var** is accepted to be processed before the end of the current event or before the next **Conditional-suffix** control element.

**DTD Content Rule**

```
&lt;TEXT&gt;
```

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Emit-space-before | Choice    | Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  - Emit-space-before: Emit a space before emitting any content text.  
  - No-space-before: Do not emit a space before emitting any content text. |
| Emit-space-after    | Choice    | Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  - Emit-space-after: Emit a space after emitting any content text.  
  - No-space-after: Do not emit a space after emitting any content text. |
| Emit-newline-before  | Choice    | Required—Since XML strips the first/last space and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  - Emit-newline-before: Emit a newline before emitting any content text.  
  - No-newline-before: Do not emit a newline before emitting any content text. |
| Emit-newline-after   | Choice    | Required—Since XML strips the first/last space and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  - Emit-newline-after: Emit a newline after emitting any content text.  
  - No-newline-after: Do not emit a newline after emitting any content text. |
### Conditional-suffix

Emits the contained text if the preceding **Conditional-prefix** within the current event was emitted.

**DTD Content Rule**

```<TEXT>`
```

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Emit-space-before   | Choice     | Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  * **Emit-space-before**: Emit a space before emitting any content text.  
  * **No-space-before**: Do not emit a space before emitting any content text. |
| Emit-space-after    | Choice     | Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  * **Emit-space-after**: Emit a space after emitting any content text.  
  * **No-space-after**: Do not emit a space after emitting any content text. |
| Emit-newline-before | Choice     | Required—Since XML strips the first/last space in each element and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  * **Emit-newline-before**: Emit a newline before emitting any content text.  
  * **No-newline-before**: Do not emit a newline before emitting any content text. |
| Emit-newline-after  | Choice     | Required—Since XML strips the first/last space and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  * **Emit-newline-after**: Emit a newline after emitting any content text.  
  * **No-newline-after**: Do not emit a newline after emitting any content text. |
Comment

Does no processing. Provided to allow documentation or notes to be included in the Mapping Table.

**DTD Content Rule**

```
<TEXT/>
```

**Emit-string**

Emits the text contained in this Mapping Table element.

**DTD Content Rule**

```
<TEXT/>
```

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Emit-space-before**     | Choice  | Required—Since XML strips the first/last space and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  • Emit-space-before: Emit a space before emitting any content text.  
  • No-space-before: Do not emit a space before emitting any content text. |
| **Emit-space-after**      | Choice  | Required—Since XML strips the first/last space and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  • Emit-space-after: Emit a space after emitting any content text.  
  • No-space-after: Do not emit a space after emitting any content text.  |
| **Emit-newline-before**   | Choice  | Required—Since XML strips the first/last space and most newlines from the parsed result, it is necessary to have this set of flags to control explicit insertion of these control codes. One of:  
  • Emit-newline-before: Emit a newline before emitting any content text.  
  • No-newline-before: Do not emit a newline before emitting any content text.  |
### Proc-doc-text

Emits the text contained in the current Structural Element.

**DTD Content Rule**

 void?

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>do-br-substitution</td>
<td>Choice</td>
<td>Required. One of:</td>
</tr>
<tr>
<td>do-br-substitution</td>
<td></td>
<td>Emit a <code>&lt;BR&gt;</code> for every newline found in the doc text.</td>
</tr>
<tr>
<td>do-xml-br-substitution</td>
<td></td>
<td>Emit a <code>&lt;br /&gt;</code> for every newline found in the doc text.</td>
</tr>
<tr>
<td>no-substitution</td>
<td></td>
<td>Disregard newlines in doc text.</td>
</tr>
</tbody>
</table>

### Proc-string

If the data cached by the containing **Proc-var** directive is a string or an atom, emits the text content of the string or a text representation of the atom’s name.

**DTD Content Rule**

 void?
Proc-integer

If the data cached by the containing Proc-var directive is an Int32 or a Uns32, emits the text representation of the value. This value is scaled using the attributes of this directive:

1. The original value is multiplied by the value of the Mul attribute.
2. The value of the Add attribute is added to the result of step 1.
3. The result of step 2 is divided by Div and the fraction is discarded.
4. The result of step 3 is converted to a string.

DTD Content Rule

Void?

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul</td>
<td>String</td>
<td>Optional. Default is 1.</td>
</tr>
<tr>
<td>Add</td>
<td>String</td>
<td>Optional. Default is 0.</td>
</tr>
<tr>
<td>Div</td>
<td>String</td>
<td>Optional. Default is 1.</td>
</tr>
</tbody>
</table>
Proc-hex

If the data cached by the containing `Proc-var` directive is an Int32, an Uns32, or a Fixed, emits the text representation of the integer portion of the value, after the scaling algorithm is applied. This value is scaled using the attributes of this directive:

1. The original value is multiplied by the value of the `Mul` attribute.
2. The value of the `Add` attribute is added to the result of step 1.
3. The result of step 2 is divided by `Div` and the fraction is discarded.
4. The result of step 3 is converted to a string.

**DTD Content Rule**

```xml
Void?
```

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul</td>
<td>String</td>
<td>Optional. Default is 1.</td>
</tr>
<tr>
<td>Add</td>
<td>String</td>
<td>Optional. Default is 0.</td>
</tr>
<tr>
<td>Div</td>
<td>String</td>
<td>Optional. Default is 1.</td>
</tr>
<tr>
<td>Num-digits</td>
<td>String</td>
<td>Optional. Default is 2.</td>
</tr>
</tbody>
</table>
**Proc-fixed**

If the data cached by the containing `Proc-var` directive is a FixedPoint number, emits the text representation of the value. This value is scaled using the attributes of this directive:

1. The original value is multiplied by the value of the `Mul` attribute.
2. The value of the `Add` attribute is added to the result of step 1.
3. The result of step 2 is divided by `Div`.
4. The result of step 3 is converted to a string. `Frac-len` controls the number of digits to the right of the decimal point. `Frac-dlm` is the fraction-radix character to be issued if `Frac-len` is greater than 0.

`Proc-fixed`, `Proc-length`, and `Proc-pixels` vary only in the default values for `Mul`, `Div`, and `Add`.

**DTD Content Rule**

Void?

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul</td>
<td>String</td>
<td>Optional. Default is 1.</td>
</tr>
<tr>
<td>Add</td>
<td>String</td>
<td>Optional. Default is 0.</td>
</tr>
<tr>
<td>Div</td>
<td>String</td>
<td>Optional. Default is 1.</td>
</tr>
<tr>
<td>Frac-len</td>
<td>String</td>
<td>Optional. Default is 2.</td>
</tr>
<tr>
<td>Frac-dlm</td>
<td>String</td>
<td>Optional. Default is &quot;.&quot;</td>
</tr>
</tbody>
</table>
Proc-length

If the data cached by the containing Proc-var directive is a FixedPoint number, emits the text representation of the value. This value is scaled using the attributes of this directive:

1. The original value is multiplied by the value of the Mul attribute.
2. The value of the Add attribute is added to the result of step 1.
3. The result of step 2 is divided by Div.
4. The result of step 3 is converted to a string. Frac-len controls the number of digits to the right of the decimal point. Frac-dlm is the fraction-radix character to be issued if Frac-len is greater than 0.

Proc-fixed, Proc-length, and Proc-pixels vary only in the default values for Mul, Div, and Add.

DTD Content Rule

Void?

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul</td>
<td>String</td>
<td>Optional. Default is 72.</td>
</tr>
<tr>
<td>Add</td>
<td>String</td>
<td>Optional. Default is 0.</td>
</tr>
<tr>
<td>Div</td>
<td>String</td>
<td>Optional. Default is 72.</td>
</tr>
<tr>
<td>Frac-len</td>
<td>String</td>
<td>Optional. Default is 2.</td>
</tr>
<tr>
<td>Frac-dlm</td>
<td>String</td>
<td>Optional. Default is “.”</td>
</tr>
</tbody>
</table>
**Proc-pixels**

If the data cached by the containing **Proc-var** directive is a FixedPoint number, emits the text representation of the value. This value is scaled using the attributes of this directive:

1. The original value is multiplied by the value of the **Mul** attribute.
2. The value of the **Add** attribute is added to the result of step 1.
3. The result of step 2 is divided by **Div**.
4. The result of step 3 is converted to a string. **Frac-len** controls the number of digits to the right of the decimal point. **Frac-dlm** is the fraction-radix character to be issued if **Frac-len** is greater than 0.

**Proc-fixed**, **Proc-length**, and **Proc-pixels** vary only in the default values for **Mul**, **Div**, and **Add**.

**DTD Content Rule**

Void?

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul</td>
<td>String</td>
<td>Optional. Default is 96.</td>
</tr>
<tr>
<td>Add</td>
<td>String</td>
<td>Optional. Default is 36.</td>
</tr>
<tr>
<td>Div</td>
<td>String</td>
<td>Optional. Default is 72.</td>
</tr>
<tr>
<td>Frac-len</td>
<td>String</td>
<td>Optional. Default is 0.</td>
</tr>
<tr>
<td>Frac-dlm</td>
<td>String</td>
<td>Optional. Default is <code>'</code></td>
</tr>
</tbody>
</table>
**Proc-enum**

If the data cached by the containing `Proc-var` directive is a string or an atom, searches for a match among the `proc-enum` choice elements that are children of this control element. If a match is found, issues the `Value-out` value of the matching `Proc-enum-choice` directive as a string.

**DTD Content Rule**

```xml
Proc-enum-choice+
```

**Proc-enum-choice**

Specifies the choice and output values for a `Proc-enum` directive.

**DTD Content Rule**

```xml
Void?
```

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value-in</td>
<td>String</td>
<td>Required—This value is compared to the value cached by the containing <code>proc-var</code> directive.</td>
</tr>
<tr>
<td>Value-out</td>
<td>String</td>
<td>Required—This value is emitted as a string if a match against <code>Value-in</code> is found.</td>
</tr>
</tbody>
</table>

**Proc-graphic-content**

Processes the content of the current structural element as a vector graphic.

**DTD Content Rule**

```xml
Void?
```

**Proc-image-content**

Processes the content of the current structural element as a bitmapped graphic.

**DTD Content Rule**

```xml
Void?
```
Void

This node is used to avoid the \texttt{<empty/>} syntax of XML and force the \texttt{<name></name>} syntax of SGML (this allows editing on any SGML editor as well as any XML editor).

Many of the above elements have the content rule "Void?". However, the Void element should never be specified, thereby leaving the containing node empty.

**DTD Content Rule**

\texttt{<EMPTY>}
